Lesson 09

数据表达

数据型语言

什么是

一门语言，不是为了执行，而是为了表达数据

有哪些

XML

JSON

YAML

样例

XML：

```xml
<Circle>
    <origin>
        <Point x="3" y="4" />
    </origin>
    <radius>2</radius>
</Circle>
```

JSON：

```json
{
    "origin": {
        "x": 3,
        "y": 4
    },
    "radius": 2
}
```

YAML：

```yaml
circle:
    origin:
        x: 3
        y: 4
    radius: 2
```

通用特性

定义了数据的大概格式，

但没有定义数据的内容

收益

使用前：

自定义文件格式，需要自己进行字符串的处理

处理字符串后，提取到单元化的数据后

再转化为内存中的业务对象

使用后：

使用语法要求的文件格式，自定义数据结构

由数据语言提供的解析器，将字符串变为 单元化的数据

自己将单元化的数据，转为内存中的业务对象

JSON

JavaScript Object Notation

是什么

JS 的字面量语法 用于描述数据

基础元素

字符串：

```
"zzax.io"
"z"
'zzax.io'
'z'
```

数值：

```
9917
99.17
```

布尔值：

```
true
false
```

空：

```
null
```

数组

```
[6, 7, 3, 8, 4]
```

数组没必要类型一致：

```
[6, 7, true, "zzax", 4]
```

对象

其实就是 Map

```
{"name1": "value1", "name2": "value2"}
```

也可以这样

```
{
    "name1": "value1",
    "name2": "value2"
}
```

key 必须为字符串，但值没必要类型一致：

```
{
    "time": "Sun Mar 17 17:40:59 EDT 2019",
    "status": "in transit",
    "code": 412
}
```

嵌套

任何的值的位置，可以使用基础元素，或者继续展开为数组/对象

淘宝买家信息 API 结果

```
{
    "user_buyer_get_response":{
        "user":{
            "nick":"hz0799",
            "sex":"m",
            "avatar":"http:\/\/assets.taobaocdn.com\/app\/sns\/img\/default\/avatar-120.png",
            "open_uid":"324324324"
        }
    }
}
```

根节点

根节点必须为 数组 或者 对象

对象作为根节点：

```
{
    "x": 3,
    "y": 4
}
```

数组作为根节点：

```
[
    3,
    4
]
```

基础元素作为根节点：

错！

```
! 3
```

GSON 与 JSON 序列化

搭建对象

```
JsonObject jsonObject = new JsonObject();
jsonObject.addProperty("x", 3);
jsonObject.addProperty("y", 4);

System.out.println(new Gson().toJson(jsonObject));
```

格式化打印

```
JsonObject jsonObject = new JsonObject();
jsonObject.addProperty("x", 3);
jsonObject.addProperty("y", 4);

Gson gson = new GsonBuilder().setPrettyPrinting().create();
System.out.println(gson.toJson(jsonObject));
```

解析

```
String json = "{'x': 3, 'y': 4}";
JsonObject jsonObject = new Gson().fromJson(json, JsonObject.class);
System.out.println(jsonObject);
System.out.println(jsonObject.get("x"));
```

JSON 作为中介的序列化

序列化：

业务对象 > JSON 对象 > JSON String

```
//  业务对象
Point point = new Point(3, 4);

//  业务对象 -> JSON 对象
JsonObject pointJson = new JsonObject();
pointJson.addProperty("x", point.x);
pointJson.addProperty("y", point.y);

//  JSON 对象 -> JSON 字符串
String json = new Gson().toJson(pointJson);

System.out.println(json);
```

反序列化：

JSON String > JSON 对象 > 业务对象

```java
//  JSON String
String json = "{'x': 3, 'y': 4}";

//  JSON String -> JSON 对象
JsonObject pointJson = new Gson().fromJson(json, JsonObject.class);

//  JSON 对象 -> 业务对象
Point point = new Point();
point.x = pointJson.get("x").getAsInt();
point.y = pointJson.get("y").getAsInt();
```

JSON 作为中介的序列化 全自动

序列化：

```java
//  业务对象
Point point = new Point(3, 4);

//  业务对象 -> JSON 字符串
String json = new Gson().toJson(point);

System.out.println(json);
```

反序列化：

```java
//  JSON 字符串
String json = "{'x': 3, 'y': 4}";

//  JSON 字符串 -> 业务对象
Point point = new Gson().fromJson(json, Point.class);

System.out.println(point);
```

XML

节点

```xml
<node prop1="value1" prop2="value2">
    content
</node>
```

属性必须使用 双引号 括起来

空节点

```
<node prop1="value1" prop2="value2" />
```

后开先关

后开的必须先关，否则错误

```
<a><b></a></b>
```

唯一根节点

不能有 并列的根节点

```
<a></a>
<b></b>
```

注释

```
<!-- zzax -->
<lesson></lesson>
```

Entities

```
<lesson>&lt;</lesson>
```

Jackson 处理 XML

序列化：

```
//  业务对象
Point point = new Point(3, 4);

//  业务对象 -> XML 字符串
String xml = new XmlMapper().writeValueAsString(point);

System.out.println(xml);
```

反序列化：

```
//  XML 字符串
String xml = "<Point><x>3</x><y>4</y></Point>";

//  XML 字符串 -> 业务对象
Point point = new XmlMapper().readValue(xml, Point.class);

System.out.println(point);
```

Base64

```
byte[] bytes = FileUtils.readFileToByteArray(new File("logo.png"));

String hexString = Hex.encodeHexString(bytes);
System.out.println(hexString);
System.out.println(hexString.length());

String base64String = Base64.encode(bytes);
System.out.println(base64String);
System.out.println(base64String.length());
```